UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

ORACLE AMERICA, INC. Plaintiff, v. GOOGLE, INC. Defendant.	Case No. 3:10-cv-03561-WHA

REPLY EXPERT REPORT OF DR. OWEN ASTRACHAN

HIGHLY CONFIDENTIAL – ATTORNEYS' EYES ONLY
INFORMATION SUBJECT TO PROTECTIVE ORDER

TABLE OF CONTENTS

I. INTRODUCTION

A. Background

1. Related to my engagement on this matter, Google has asked me to review and respond to various technical issues in the fair use and rebuttal expert reports submitted on behalf of Oracle America, Inc.

2. As at the time of my prior reports, I am being compensated for my work in this litigation at the rate of $500 an hour, and my compensation does not depend in any way on the outcome of this litigation.

3. At this time, I have not created any exhibits to be used as a summary of, or as support for, my opinions. I reserve the right to create any additional summaries, tutorials, demonstrations, charts, drawings, tables, and/or animations that may be appropriate to supplement and demonstrate my opinions if I am asked to testify at trial.

4. I understand that discovery is ongoing in this case. I therefore reserve the right to supplement my opinions after I have had the opportunity to review deposition testimony or in light of additional documents that may be brought to my attention. Further, if Oracle or its experts change their opinions (either explicitly or implicitly) in such a manner as to affect my conclusions, I may supplement my opinions.

B. Professional qualifications

5. A summary of my professional qualifications was provided at paragraphs 5-9 of my Opening Expert Report Of Dr. Owen Astrachan On Technical Issues Relating To Fair Use (Jan. 8, 2016) (Opening Report), as well as Exhibit A attached thereto.

C. Documents and information considered

6. My opinions are based on my relevant knowledge and experience, the trial testimony and exhibits from the first phase of the original trial (which I attended daily), and the

materials cited and discussed in this reply report, as well as the materials cited and discussed in my opening report (including those listed in Exhibit B) and my rebuttal report (including those listed in Appendix A). Additional documentation and information considered and/or relied upon is identified in Appendix A to this reply report.

7. I also have specialized knowledge of the matters set forth herein, and if called as a witness I could and would testify competently thereto.

8. My research and analysis of the materials, documents, allegations, and other facts in this case are ongoing, and if additional information becomes available through discovery and depositions in this action, I reserve the right to provide supplemental opinions.

II. TRANSFORMATIVENESS

A. Android is a transformative platform

9. Oracle's experts focus on the assertion that the 37 Java SE API packages are supposedly used in the same way in Android as they are in Java SE 5. See, for example, Schmidt Rebuttal Report ¶ 50. The declaring code in Android does serve the purpose of allowing access to specific functionality. But that is not the point. The declaring code was made a part of a larger structure that gave it a different meaning or expression and thus transformed it; as an example, a thumbnail picture displayed by a search engine still performs the function of showing you what the picture looks like, but plays a different role when it is used as part of search results. Android put the 37 Java SE API packages into a structure where they could be used as part of a full-stack smartphone operating system, something that those 37 Java SE API packages had never been a part of before.

10. Android went beyond putting the declaring code into a different overall structure that gave it different meaning or expression. Android also transformed the declaring code by associating it with new implementing code that was optimized for use in the new environment of

a full-stack mobile platform. As Dan Bornstein testified in the first trial, Google edited and modified the Android implementing code to make it "work better in the environment of Android." RT 1818:17-1819:3; *see also* Lee 8/3/11 deposition tr. at 71:13-24. In many instances, implementing code was written and optimized to save the space that is critical for a smaller mobile device by allowing the reuse of code that would be present elsewhere in the software, to create the security enhancements necessary for an effective mobile platform, or to make the outputs more useful for a mobile device, as I explain in more detail in the following paragraph. And, despite language in the reports of Oracle's experts that might mislead a reader into believing that Android used pre-written code from Sun/Oracle to implement these APIs (see, for example, Schmidt Rebuttal Report ¶ 12; *id.* ¶ 31; *id.* ¶ 235; Kemerer Rebuttal Report ¶ 51; *id.* ¶ 152), based on my experiences in and leading up to the first trial, the record is undisputed that aside from a handful of lines of implementing code, Google used only the declaring code and SSO of the 37 Java SE API packages while writing its own implementing code. I also note, as I did in my opening report at paragraph 250, that using only the declaring code and SSO from the 37 Java SE API packages is using a minimal amount of code from those packages in a way that comports with the free and open nature of the Java programming language.

11. I note four examples of improvements that were made in the implementing code for the 37 Java SE API packages to illustrate the transformation enabled by the implementing code in Android. In the Android implementation of the classes in the java.text package, IBM's implementation of the ICU - International Components for Unicode (http://site.icu-project.org) is used in 11 classes as part of ensuring support for internationalization that is missing from the Sun/Oracle implementation of the same classes in Java SE 5. This support for internationalization is important for mobile devices that may be sold in different countries or

taken to different countries. In the java.security package for Android, the implementation of the MD5 cryptographic hash algorithm is based on the OpenSSL libraries that are used with the Android platform because they provide the speed and efficiency that are important for a mobile platform. A comparison of the Android implementing code for the TimeZone class in java.util is roughly 56% of the size of the implementing code in the Java SE 5 implementation, as measured by lines of code, again saving space that is critical to mobile devices. Finally, the Android implementing code in the java.net package makes use of BlockGuard classes that are part of the Android stack to help ensure that the user interface for mobile devices remain responsive. Java SE 5 does not have such functionality.

12. Oracle's expert reports constantly refer to different pre-existing operating systems or platforms as "Java-based." See, for example, Schmidt Rebuttal Report ¶ 35 (referring to an "array of widely available Java-based embedded and mobile devices"); Jaffe Rebuttal Report ¶ 118 (referring to "Java-based mobile platforms including BlackBerry and Symbian"); *id.* ¶ 122 (claiming that "Java-based phone adoptions had grown to 85% of all phones shipped at the time" by 2007); *id.* ¶ 126 (referring to the "Java-based Symbian OS"); *see also* Schmidt Rebuttal Report ¶ 40 (referring to "Java-enabled operating systems"). The criteria that Oracle's experts use to determine that something is "Java-based" are unclear, but they do not appear to be accurate. Symbian OS was not a "Java-based" OS. Its primary application programming language was C++ or QML, and it secondarily supported other programming languages, including an ability to use Java ME. Moreover, Oracle's experts appear to refer to a device capable of using Java ME as "Java-based." *See* Schmidt Rebuttal Report ¶ 41 ("The Java Micro Edition ('Java ME') platform was initially the version of Java used to power mobile and embedded Java devices starting around 2001."); *id.* ¶ 44 ("[T]he Blackberry 5810, was released

in 2002 and was based on Java ME"). I note that Java ME is not the copyrighted work at issue here; indeed, Java ME has only a fraction of the 37 Java SE API packages at issue in this case as well as having additional APIs of its own not found in Java SE 5, *see* Kemerer Opening Report Appendix S, and, as a result, cannot have the SSO of the 37 Java SE API packages that are at issue in the case. I therefore disagree that devices that are "Java-based" but have not been shown to use the declaring code and the SSO of the 37 Java SE API packages are relevant to this case, which is about alleged infringement of the declaring code and SSO of the 37 Java SE API packages in Java SE.

13. Oracle's contention that SavaJe supposedly showed that the full Java SE stack could be run on a mobile phone (Schmidt Rebuttal Report ¶ 43) only highlights the fact that the vast majority of the "Java-based" devices that its experts discuss do not use the Java SE platform that is the copyrighted work at issue in this case. Again, to the extent that Oracle points to devices running Java ME, such devices cannot have used the SSO of the 37 Java SE API packages, because Java ME does not have all 37 of those packages. Android is transformative: by taking the 37 Java SE API packages out of the context of Java SE and putting them into a new context optimized for use with mobile devices, it creates a new and very different platform in a different market. I explained this transformation in paragraphs 151 through 167 of my rebuttal report, where I explored the workings of Android's Dalvik virtual machine and the new Android Runtime (ART), which were both designed specifically for use in a mobile platform, as well as in paragraphs 229 through 255 of my opening report, where I discussed differences in the implementing code used by the two platforms. Android also enabled the creation of something that Java developers had not seen before: an online marketplace for applications to run on smartphones, akin to what Apple had done for Objective-C developers with its App Store.

14. Moreover, putting the entirety of Java SE (which is designed for use in server and desktop systems) on a phone is not a good use of memory or computing power, as demonstrated by a contemporaneous review of the Jasper S20 phone. *See* http://gizmodo.com/210575/jasper-s20-cellphone-rocks-the-java-os-people-flee-in-fear. The article starts off by asking rhetorically "Lots of people hate Java for being both slow and a memory hog, so what better place to put it than on a mobile phone?" *Id.* This emphasizes the problems that Java had running on mobile phones before the release of the transformative Android. As the author put it, he did not like the idea of Java on a mobile phone because "I don't want to hit 'up' and have to wait for three seconds in order to see a response." *Id.* The different implementing code that Android provided for the 37 Java SE API packages, as well as the improved overall structure in which they were placed, solved these sorts of problems with Java SE. And it is no surprise that Oracle fails to point to any evidence that the SavaJe device made a ripple in the market: another contemporaneous review of the Jasper S20 notes that "if you're a Java developer or you just like dated tech running a relatively untested software platform, the Jasper S20 might be the phone for you." http://www.engadget.com/2006/05/13/savaje-releases-jasper-s20-java-phone/.

15. Although Oracle's experts speak of major Java mobile technology plans on the part of Sun, Schmidt Rebuttal Report ¶ 45, and claim that JavaOS supposedly was designed to run in devices including mobile phones, Schmidt Rebuttal Report ¶ 46, the testimony at the first trial demonstrated conclusively that Sun was never able to bring a mobile operating system or device based on Java (including JavaOS) to market, and it took the innovation of Android to use the 37 Java SE API packages in a transformative manner that made them useful in a modern smartphone platform. Indeed, as noted in paragraph 151 of my opening report, Oracle itself is

now attempting to use the Android platform to provide its OpenJDK software, showing once again that Oracle is not able to bring Java to a modern smartphone on its own.

B. Oracle's experts mischaracterize Android

16. Dr. Schmidt claims that "Google's use in Android is not transformative" because it is "intended to be used by software developers to create software that will be run by a virtual machine." Schmidt Rebuttal Report ¶ 36. As explained in my rebuttal report and as discussed above, the Android operating system no longer relies on a virtual machine; instead, the ART runtime employs ahead of time compilation to optimize the user experience. And even when Android was still using the Dalvik virtual machine, that virtual machine was optimized for mobile devices in a way in which the Java virtual machine never has been.

17. Dr. Schmidt asserts that the Android Native Development Kit ("NDK") was "not novel or transformative" because Java purportedly provided the same capability through its Java Native Interface ("JNI"). Schmidt Rebuttal Report ¶ 53. The NDK represents a transformative use because it provides access to a full-stack mobile operating system when that is needed rather than simply to numeric, graphic, cryptographic or other routines, as is the case with the JNI in Java SE 5. Because Android developers have a common operating system and supporting libraries on all Android devices, the use of the NDK in Android is another example of the transformation enabled by Android. Moreover, Dr. Schmidt's position on the use of JNI to make calls to native routines is difficult to reconcile with his position on "Write Once, Run Anywhere," because to the extent that Java makes calls to native libraries, the code would need to be adapted to the specific hardware associated with those libraries.

III. JAVA CLASSES

18. Dr. Schmidt criticizes my choice of methods from the java.lang.Math class to explain the workings of Java, calling it "suspect." Schmidt Rebuttal Report ¶ 81. In my opinion,

java.lang.Math serves to illustrate well the manner in which the Java programming language works. In fact, it is such a useful illustration of the workings of the Java programming language that multiple witnesses used java.lang.Math as a teaching tool at the first trial, and it was also used as an example in various opinions relating to the first trial. Accordingly, I chose to use it as a pedagogical tool in my report as well. Moreover, it is not clear whether, by making this critique of my choice to use java.lang.Math, Dr. Schmidt means to suggest that some Java classes are less subject to principles of fair use than others.

19. Dr. Schmidt goes on to explain concepts relating to classes, interfaces, and inheritance. Schmidt Rebuttal Report ¶ 92. Although I do not disagree with his general description of how these concepts work, I do note that they are common features of object-oriented programming languages, not concepts that were developed as part of, or unique to, the Java programming language.

IV. THE FUNCTIONAL NATURE OF THE 37 JAVA SE API PACKAGES

A. Dr. Schmidt's analysis concedes that the Java programming language requires certain characteristics

20. Dr. Schmidt claims that "there are numerous ways that a platform developer could create comparable functionality, but choose a different SSO." Schmidt Rebuttal Report ¶ 94. He then gives an example of "duplicat[ing] the complete functionality of every class within the Java API, but without using inheritance (beyond the implicit inheritance of class Object)." *Id.* His recognition of the fact that even in his proposed alternative embodiment there would still be an implicit inheritance of class Object shows that certain characteristics of the Java programming language, such as the fact that all objects implicitly inherit from the class Object, must be part of any implementation using the Java programming language.

B. Dr. Schmidt's analysis shows that the 37 Java SE API packages are functional

21. Dr. Schmidt gives another possible hypothetical alternative to using the 37 Java SE API packages: "to duplicate the complete functionality of every class within the Java API, organized into an arbitrarily large and complex inheritance tree without duplicating the SSO of the Java SE platform." *Id.* He concedes, however, that "[n]either of these are particularly attractive options." *Id.*

22. Dr. Schmidt's opinion that his proposals are not attractive is not based on the aesthetic value of those proposals. Instead, it is based on how well such a system would function in carrying out its purpose. This demonstrates that the SSO primarily reflects a fundamental aspect of the Java programming language that is highly functional, and will affect the functional ability of programmers to use the Java programming language if not constructed in a suitable fashion.

23. Moreover, Dr. Schmidt concedes that at least some of the 37 Java SE API packages are necessary to the Java programming language, which demonstrates that those API packages are functionally necessary for compatibility with the language. He states in his rebuttal report that "Android Java copied ten times more classes than strictly necessary," Schmidt Rebuttal Report ¶ 227, which reflects his understanding that at least some classes are "necessary" for compatibility with the Java programming language. He further states that "not all of the copied declarations represent functional requirements of the language," *id.* ¶ 240, implicitly recognizing that at least some of the 37 Java SE API packages *are* functional requirements of the language. *See also id.* ¶ 240 (referring to the "minimum amount necessary"); ¶ 246 (referring to the "minimum functional requirement"). Something that is necessary for compatibility is by its nature functional.

C. The visualization that Dr. Schmidt provides further shows why the 37 Java SE API packages are necessary for basic compatibility with the Java programming language

24. Dr. Schmidt provides what purport to be visualizations of the SSO of Java SE 5. *See* Schmidt Rebuttal Report Figures 11-15. These visualizations bear no resemblance to the way that Sun/Oracle shows the SSO outside of the context of litigation, where it is depicted as part of a hierarchical structure. *See* TX 1028. With respect to Trial Exhibit 1028, Dr. Reinhold testified at the first trial: "Developers use this poster," RT 598:19-20, and it was "not created for this litigation." RT 598:24-599:3. Neither of those things is true about these figures from Dr. Schmidt's report. Leaving that aside, however, Dr. Schmidt's litigation-driven visualizations simply reflect the fact that the 37 Java SE API packages are integral to the Java programming language and accordingly necessary to make effective use of the Java programming language in the context of a mobile platform.

D. Dr. Schmidt does not dispute that Java borrowed some of its APIs from pre-existing APIs

25. Dr. Schmidt claims that Google could have used declaring code similar to that found in the ObjectSpace Java Generics Library ("JGL"). *See* Schmidt Rebuttal Report ¶ 118. However, his Table 7 illustrates that Java itself copied multiple method names from JGL, doing little more than changing the package in which they are found to "Collections." For example, "Copying.copy()" in JGL became "Collections.copy()" in Java; "Fill.fill()" in JGL became "Collections.fill()" in Java; "Reversing.reverse()" in JGL became "Collections.reverse()" in Java; and "Sorting.sort()" in JGL became "Collections.sort()" in Java. This proves my point about Java using common names developed by those who had come before Java. This should come as no surprise; as noted in paragraph 204 of my opening report, James Gosling, one of the creators of the Java programming language, said that they tried to borrow such things from existing

programming languages. Dr. Schmidt also points out that Java copied only some, but not all, of its java.lang.Math class methods from C++. *See* Schmidt Rebuttal Report ¶ 173. Of course, Dr. Schmidt also acknowledges, as he must, that Android also did not take all of its declaring code and SSO from Java either, so Java is no different in using some but not all of the material at issue. Dr. Schmidt further argues that "the duplication of copyrighted SSO, names, and API is completely different from using industry standard compliant names and syntax." *Id.* ¶ 176. Aside from presenting no evidence that the POSIX material to which he refers is not copyrighted in the same manner as Java SE 5, his argument also begs the question of whether the 37 Java SE API packages are themselves an industry standard.

26. Dr. Schmidt concedes that both Java and C++ use the printf() API, as I explained in my opening report. Schmidt Rebuttal Report ¶ 171. He then goes on to argue that the fact that Java also has the System.out.println() function somehow changes the fact of Java's taking of APIs first created by others. It does not. Though System.out.println() can also be used for printing in Java, that does not alter the fact that the printf() functionality works in the same way in both Java and C++.

V. COMPATIBILITY

A. Android is compatible with the Java programming language

27. Dr. Schmidt again conflates or confuses the Java programming language and the Java SE Platform in his rebuttal report. Despite acknowledging that the compatibility I discussed in my opening report was compatibility with the Java programming language, he goes on to say that "both Android's runtime environment and virtual machine are incompatible with Java SE." Schmidt Rebuttal Report ¶ 204; *id.* ¶ 248. Given the degree to which Google transformed the 37 Java SE API packages, Google has never claimed that the Android runtime was compatible with Java SE. And as discussed in detail in my rebuttal report, Google has never claimed that

Android was compatible with, or implemented, any particular Java specification. Because Google does not make such a claim, there is no reason for it to run any Java Compatibility Kit, and it does not hold Android out as being an implementation of any Java specification. I also note that Dr. Schmidt's position here is odd: he appears to be effectively claiming that the problem with Android is not that it copies the SSO of 37 Java SE API packages, but that it did not copy the SSO of *all* of the API packages in Java SE.

28. Moreover, as noted above, Dr. Schmidt concedes that at least some of the 37 Java SE API packages are functionally necessary for compatibility with the Java programming language.

B. Compatibility with Java specifications is not a requirement

29. Dr. Schmidt once more appeals to the "philosophy" of "Write Once, Run Anywhere," Schmidt Rebuttal Report ¶ 200. As I discussed in paragraphs 130 through 137 of my rebuttal report, compatibility is a philosophy and not an actual requirement of Java, given the manner in which Sun/Oracle itself has fragmented the platforms that it offers.

30. More importantly, however, Dr. Schmidt recognizes that the manner in which Sun/Oracle has chosen to license OpenJDK "relieves [users of OpenJDK] of compatibility requirements." *Id.* Accordingly, Dr. Schmidt has no basis for any continued insistence that compatibility with a particular Java specification is absolutely required by Sun/Oracle.

VI. USE OF APIS

A. Google does not seek to copyright the declaring code and SSO of its APIs

31. Dr. Schmidt engages in a lengthy discussion of Google's agreements relating to its own APIs. Although he claims that certain of Google's agreements "specifically reemphasize the presence and extent of Google's copyright," Schmidt Rebuttal Report ¶ 190, I disagree. Based on my review of the material cited in the footnote, I do not see any such emphasis. To the

contrary, the references to copyright in the material relate to the underlying product or services provided by way of the API.

32. More importantly, I do not see any mention by Dr. Schmidt of claims to copyright covering the declaring code and SSO of Google's APIs. Despite recognizing elsewhere in his report that the declaring code and SSO are the part of Oracle's work that has allegedly been infringed, *see, e.g.*, Schmidt Rebuttal Report ¶ 61, Dr. Schmidt does not focus on the declaring code or SSO of Google's APIs. Instead, he refers broadly to Google APIs that are allegedly "very similar to the APIs at issue." *Id.* ¶ 208.

B. Google's agreements relate to the use of its underlying services, not the declaring code and SSO of its APIs

33. Dr. Schmidt thereafter discusses contractual terms of service to which Google requires users of its products and services to agree in return for the ability to use the product or service. *See* Schmidt Rebuttal Report ¶ 208. The terms of the agreements to which he refers show that Google is not seeking to control the APIs themselves, much less their declaring code or SSO, but rather the services, implementing code, and data that may be accessed by way of the APIs. This way of using the term "API" is very different than the way that Dr. Schmidt was previously using it. *See id.* (noting that using the API means use to "*access Google servers* through the AdWords API," "*send information to AdWords accounts* using an AdWords API Client," or "*receive information from Google* in response to AdWords API calls" (emphases added)); *see also id.* ¶ 211 (noting that the AdWords APIs are used "to obtain access to Google's backend software implementations and data relating to Google's search services"); *id.* ("A developer would type the API declaring code in their client programs, *in order to cause software implementations within Google's services to carry out desired functionality*." (emphasis added)). Regardless of whether declaring code is employed to use these services, it is clear that what

Google is protecting is the underlying functionality and data, not the declaring code or SSO of the APIs. Dr. Kemerer makes much the same argument, referencing the same materials, in paragraphs 220 and 221 of his rebuttal report. In my opinion, agreements relating to access to Google data and the ability to use Google implementing code have no bearing on Google's use of the declaring code and SSO of the 37 Java SE API packages here, which do not call implementing code or data developed by Oracle.

34. Dr. Kemerer's similar discussion of Google agreements relating to Android fares no better. He asserts that these are "agreements to control the APIs," Kemerer Rebuttal Report ¶ 210, but admits that the agreements are concerned with benefits that Google provides through the use of its own implementing code and trademarks, such as the ability to "distribute Google applications, use the Android branding or trademarks, or gain early access to the Android source code." *Id.* ¶ 209. This is likewise true of his discussion of an article regarding APIs from Google Play Services. *See id.* ¶ 214. That article notes the need to "assess the intrinsic value of the data or services that an API makes available," which differs from the API itself.

35. Dr. Kemerer asserts that Google imposes restrictions on the "copying" of its APIs. Kemerer Rebuttal Report ¶ 215. But nothing that he cites thereafter suggests that Google seeks to prevent others from using the declaring code or SSO of Google APIs. Instead, the materials cited by Dr. Kemerer reflect that Google charges for use of the underlying *services* that can be accessed through Google APIs. This distinction is reflected in the testimony of Urs Hölzle quoted by Dr. Kemerer in paragraph 218 of his rebuttal report: "And they had to pay you to get access to the API; right? . . . THE WITNESS: To the service, yes." It is also reflected in the Terms of Service for the Google Maps/Google Earth APIs, referenced by Dr. Kemerer in paragraph 216 of his rebuttal report, which state that the "Google Maps/Google Earth APIs are a

collection of services that allow you to include maps, geocoding, places, and other Content from Google in your web pages or applications." *See* https://developers.google.com/maps/terms.

36. Dr. Kemerer continues to argue that Google controls the use of its APIs, yet what he points to is not the APIs, but instead the "data resulting from use of the APIs," Kemerer Rebuttal Report ¶ 225, and "the data resources behind the APIs," *id.* ¶ 226. Such data is distinct from the declaring code and SSO of the Google APIs, and the protection of that data does not represent protection of the declaring code or SSO of the APIs.

37. More examples of Dr. Kemerer's confusion between the data behind APIs and the APIs themselves are found in paragraph 228 of his rebuttal report. Moreover, Dr. Kemerer does not offer the full story of some of the examples he cites. He states that "in 2015, Google sent a cease and desist letter to an Internet service called Routebuilder, that Google asserted was using the Google Maps APIs in violation of Google's agreement governing those APIs." Kemerer Rebuttal Report ¶ 228. But Dr. Kemerer does not report that shortly thereafter Google noted that the letter was a mistake and withdrew it. *See* https://medium.com/@andrewcmartin/google-is-no-longer-forcing-routebuilder-to-shut-down-8d4882f31447#.4jlgsvmu8.

38. Indeed, the materials cited by both Dr. Schmidt and Dr. Kemerer make clear that Google makes the declarations for its APIs freely available and encourages others to use them to gain easy access to Google products and services, and to use the Google-developed data provided by way of the APIs. Far from treating its API declarations as proprietary, Google allows others to use those declarations in building their own software, so long as the underlying code and data is used in an appropriate manner.

C. Dr. Kemerer seeks to sow confusion in his discussion of expectations surrounding APIs

39. Throughout the first portion of his rebuttal report, Dr. Kemerer repeatedly specifies what was allegedly copied by Google: "the declaring code and structure, sequence and organization" of the 37 Java SE API packages. Kemerer Rebuttal Report ¶ 1; *see also* Kemerer Rebuttal Report ¶¶ 13, 23, 27, 53, 58, 59, 79, 108, 119, 136, 138, and 140 (referring to the declaring code and SSO as allegedly copied by Google). Yet in the latter portion of his report that discusses supposed expectations of control over APIs, which starts with paragraph 145, Dr. Kemerer ceases to mention declaring code or SSO. Instead, he starts referring to "APIs and similar programs." Kemerer Rebuttal Report ¶ 145; *see also id.* ¶ 151 (referring to "pre-written programs that developers use to develop other programs"); *id.* ¶ 152 (referring to "APIs or similar code"); *id.* ¶ 171 (referring to "APIs and other software development tools"). From my reading of his rebuttal report, that is because he is referring to things other than the declaring code and SSO of APIs.

40. Dr. Kemerer also confuses the issues by referring to articles that relate to the potential effect of rulings in this very case rather than the historical views on the use of declaring code and SSO. *See* Kemerer Rebuttal Report ¶¶ 157, 159.

41. Dr. Kemerer points to the terms of a license Apple puts out for its XCode and Apple SDKs. *See* Kemerer Rebuttal Report ¶ 160. An expert in this field would recognize that an SDK itself is far different from the APIs to the SDK, and includes a great deal of implementing code and other proprietary materials over which companies often assert intellectual property rights.

42. Dr. Kemerer's reference to articles discussing Microsoft's control of its Windows software, *see* Kemerer Rebuttal Report ¶¶ 161-163, is likewise misdirected. The materials cited

relate to the fact that Microsoft owns the Windows software source code, and therefore is naturally able to control changes to the APIs for that software. Dr. Kemerer claims that Microsoft places restrictions on "copying and use of the APIs," *id.* ¶ 163, but does not point to any particular term of Microsoft's agreements in support of this assertion. Leaving aside that he is not focused on the declaring code and SSO of APIs, I did not see any mention of copying of APIs, just of copying of the materials returned by a search query submitted through the APIs.

43. Even further afield, Dr. Kemerer references the Unreal Engine SDK. *See* Kemerer Rebuttal Report ¶ 164. The developers who pay royalties to Epic would be using not just APIs, but the entirety of the Unreal Engine SDK, which includes its implementing code. Dr. Kemerer himself admits that the Unreal Engine includes "a set of software libraries" in addition to the APIs. In any event, the fact that companies may choose to develop proprietary APIs says nothing at all about practices relating to the use of declaring code or SSO from widely publicized and publicly specified APIs for a programming language.

44. Dr. Kemerer then refers to various "debates" regarding how to manage APIs, *see* Kemerer Rebuttal Report ¶¶ 165-167, but does not explain how the existence of these debates has anything to do with the use of the declaring code or SSO of publicly available APIs.

45. The references by Dr. Kemerer to "protection of underlying data and resources," Kemerer Rebuttal Report ¶ 182, "security of data, services and software implementations that are behind the API code," *id.* ¶ 184, and "protect[ing] the backend," *id.* ¶ 185, again show that Dr. Kemerer is not talking about the declaring code and SSO that he acknowledges is at issue here, but instead about the materials that lie beneath the APIs.

46. Dr. Kemerer's insistence that Java "inherently has always required limitations on copying and use of APIs," Kemerer Rebuttal Report ¶ 194, cannot be squared with Sun/Oracle's

actions if it is meant to suggest (as Dr. Kemerer does) that Sun/Oracle had to be able to prohibit "subsetting, supersetting, or uses that did not pass the compatibility tests provided by Sun and Oracle." As explained in paragraph 259 of my opening report, the FAQ that Sun published after its decision to release OpenJDK acknowledged that users of OpenJDK would in fact be able to create incompatible implementations of Java SE. Incompatible implementations would include those that subset or superset the API packages found in Java SE.

47. When Dr. Kemerer gets around to once again discussing the declaring code and SSO of the Java API packages in Java SE, he contends that Sun's Java Specification License "states that licensees may copy and use the declaring code and structure, sequence and organization of the Java API packages, as long as the licensee *strictly* creates only a compatible implementation using those APIs and passes the Java compatibility tests, consistent with Java's 'Write Once. Run Anywhere.' principle." Kemerer Rebuttal Report ¶ 199. I have reviewed the license that Dr. Kemerer refers to, and do not see any express mention of the copying and use of either declaring code or SSO in the license.

VII. DECLARING CODE AND IMPLEMENTING CODE

A. Oracle's copyrighted work is Java SE 5

48. Dr. Schmidt proposes to compare the amount of declaring code used in Android with the overall amount of declaring code in Java SE 5. *See* Schmidt Rebuttal Report ¶ 231. I do not see a basis for such a comparison. I understand that the relevant comparison is with the copyrighted work as a whole, and Dr. Schmidt concedes that point. *See id.* ¶ 230 ("I understand that this comparison should be made against the copyrighted work as a whole."). Here, the copyrighted work is Java SE 5. Java SE 5 is not composed of only (or even mainly) declaring code. It contains far more in the way of implementing code, which was not copied by Google, as well as a virtual machine and runtime. Accordingly, I disagree with Dr. Schmidt's suggestion

that it would be proper to judge the substantiality of copying by focusing only on declaring code and ignoring all of the implementing code and other material found in the copyrighted work at issue. Dr. Kemerer makes the same suggestion in paragraph 144 of his rebuttal report, and I disagree with him for the same reasons.

49. Oracle admits that what is at issue here is several thousand lines of code, and does not contest that the Java SE 5 code base, the copyrighted work, contains more than 2.8 million lines of code. *See* Astrachan Opening Report ¶ 140. Thus, the amount of material taken from this work represents a fraction of a percent of the code base. As explained in my opening report, this is not a "substantial" portion of the work as a whole. Dr. Kemerer does not disagree that this is a miniscule portion of the work as a whole, instead asserting that even if these few thousand lines of code are "considered only a small percentage of Java's and/or Android's overall code structure, [they] represent its critical and valuable components." Kemerer Rebuttal Report ¶ 27. Dr. Kemerer attempts to argue that this tiny amount of code is "substantial" by repeatedly attempting to equate "substantiality" with other things. He claims that this amount of code is substantial because it is allegedly "highly central," Kemerer Rebuttal Report ¶¶ 17, 18, 21, 54; because it is allegedly "important," Kemerer Rebuttal Report ¶¶ 20, 22, 23, 55, 78, 121, 135; or because it is allegedly "critical," Kemerer Rebuttal Report ¶¶ 53, 58, 59. As explained in my rebuttal report and below, the tests that Oracle's experts conduct to show the importance of the 37 Java SE API packages are flawed since the removal of any API packages will cause a failure to build. More importantly, however, Dr. Kemerer's repeated resort to such characterizations of the 37 Java SE API packages only highlights his inability to rebut my point: that the amount and substantiality of the code taken is a tiny percentage of the copyrighted work as a whole.

50. Dr. Kemerer's allegations that other programs may have consisted of only a few thousand lines of code, *see* Kemerer Rebuttal Report ¶ 134, do not make a difference. Sun/Oracle chose to copyright the entire Java SE platform as a whole, and the fact that other, very different, copyrighted programs from other contexts and eras may be smaller does not change the size or the nature of the work at issue here, especially given that those programs may not even have been written in an object-oriented language with declaring code and an SSO.

B. Implementing code is more valuable to developers than declaring code

51. Dr. Schmidt argues that "the declaring code and SSO of the 37 Java API packages are relatively more important to software developers than the underlying implementing code" because (given the nature and purpose of APIs) developers "do not need to know or be concerned about how they [sic] API packages are implemented." Schmidt Rebuttal Report ¶ 235. I respectfully disagree.

52. In my experience, developers will shy away from APIs with implementing code that cause too many bugs. Rather than continue to try programming with such APIs, developers will instead move on to something else. Obviously, a developer would most like to have an easy-to-use API atop implementing code that works flawlessly. However, if a developer had to choose between working with a counterintuitive API that nonetheless had excellent implementing code providing good results, or working with an easy-to-use API that had poor implementing code producing many bugs, it is my opinion that developers would prefer the poorly written declaring code with the well-written implementing code. It is true that they do not need to know *how* the implementing code works, but they are keenly interested in *how well* it works.

53. Dr. Kemerer claims in paragraph 144 of his rebuttal report that declaring code is "disproportionately more important than the underlying Android source code." I disagree with

his assertion on this point for the reasons set forth above, as well as in my opening and rebuttal reports where I discuss the Android framework as a whole. I further disagree with Dr. Kemerer's opinion that "developers only perceive the expressive declaring code when they are developing apps." Kemerer Rebuttal Report ¶ 256. In my experience, developers will also be aware of whether the code has a reputation for performing well, and that reputation will flow from the quality of the implementing code that it uses.

54. Moreover, a program written in Java that had no implementing code and only declaring code could not produce any useful results; it would only allow calls to be made to stubs of code that would not perform any functions.

55. Further, Dr. Kemerer's dim view of implementing code is contrary to Sun/Oracle's own business plans with respect to Java. Sun's former CEO, Jonathan Schwartz, has said that its business model was to compete with others on the implementation of the 37 Java SE API packages. RT 1962:4-5. Because everyone would use the same declaring code and SSO in such a business model, only the implementing code, and not the declaring code, could possibly provide a competitive edge. Put another way, Sun would not have tried to attract customers to use its particular implementation based on the declaring code, since that code would be identical for both Sun and those companies with which it sought to compete.

56. This is also shown by Sun/Oracle's business plans relating to the release of OpenJDK. OpenJDK contains all of the same declaring code as Java SE. Yet Sun/Oracle believed that it could nonetheless still make a business out of licensing its proprietary version of Java SE. The difference between the two implementations would have to come in the implementing code, not in the declaring code.

VIII. THE TESTS CONDUCTED BY ORACLE'S EXPERTS AS PART OF THEIR REBUTTAL REPORTS ARE NOT MEANINGFUL

A. Dr. Schmidt's build tests are not meaningful

57. Dr. Schmidt apparently seeks to rebut the evidence that certain classes are essential to compatibility with the Java programming language by showing that a version of Android with most of the 37 Java SE API packages removed but those essential classes left intact will not build. As discussed and demonstrated in my rebuttal report, generally speaking, removing any class from a complex platform such as Android will cause such a failure to build, so the fact that a failure to build occurred here is unsurprising. Indeed, it would have been startling if Android were to build with vast swathes of the classes it uses excised from the code.

58. Dr. Schmidt purports to find meaning in the fact that Android failed to build when all of the 37 Java SE API packages except for java.lang were removed. I disagree that this demonstrates anything meaningful. The fact that a particular API package is closely connected to the Java programming language does not mean that having that package alone left intact would be expected to result in a successful build.

59. The workings of Dr. Schmidt's build test as to the 61 classes that were selected by Dr. Reinhold as "tightly related to the Java programming language" are unclear. Dr. Schmidt says that he "deleted all of the 37 copied packages except for the 61 classes that are mentioned in" Trial Exhibit 1062, and then "searched for these 61 source files and delete [sic] them." Schmidt Rebuttal Report ¶ 244. Thus, it is unclear whether the 61 classes were part of the code base that Dr. Schmidt attempted to build or not. Either way, it is my opinion that Dr. Schmidt's test is not meaningful because no one would expect a system to successfully build after large amounts of source code designed to be part of the system were deleted.

B. Dr. Kemerer's "centrality" tests are flawed and serve to show at most how important the 37 Java SE API packages are to the Java programming language

60. Dr. Kemerer conducts an analysis of the connections between various classes in Android, with a particular focus on the 37 Java SE API packages.

61. Given that, as demonstrated in my rebuttal report and above, having any of the API packages removed from Android will cause a failure to build, the fact that one class or another is connected more often and/or in more ways to other classes is not particularly informative.

62. As I understand it, the PageRank system that Dr. Kemerer uses for his analysis is commonly used for determining links between things like websites to determine how popular a site is by seeing how many links it has and, in turn, how many links the linking items have. Although Dr. Kemerer references an academic paper in which PageRank has been used in conjunction with source code, *see* Kemerer Rebuttal Report at 13 n.15, PageRank does not necessarily provide a meaningful result for something like Java classes. Recent work on the use of PageRank in the context of source code provides several recommendations and caveats for using PageRank to evaluate the importance of classes in an object-oriented system. *See* http://staff.cs.upt.ro/~ioana/papers/EnaseSoraSpringer2015.pdf. In particular, this paper argues that a "model has to take into account both the strength of the dependencies and also include back-recommendations." I note that Dr. Kemerer indicates that the output of running a dependency analysis using Understand is the input to the calculation of PageRank values using NetworkX software. *See* Kemerer Rebuttal Report ¶¶ 61-63. Although he fails to indicate how the three numeric values output from Understand are used in calculating PageRank, none of the numeric values from Understand include the weights and strengths that are called for in the paper referenced above. Understand also counts dependencies in a way that is contraindicated by the

paper, by an understanding of PageRank, and by applying common understandings of how classes are dependent.

63. As an example, the output of running Understand indicates that the class java.util.ZipFile is noted as referencing the class java.util.Vector eight times. This weighting does not comport with either PageRank or the recommendations from the paper cited above. As an example, consider the definition and initialization of an instance variable inflaters:

```
private Vector inflaters = new Vector();
```

The instance variable is weighted as a three in the output of Understand: defining a variable, creating an object, and calling a constructor. Two calls of the Vector.size() method are weighted as two, and a call of .get(), .remove(), and .add() each add one. This overestimates the importance of the dependency on Vector. The paper cited above explains that by not using weighting, PageRank will give very different results than other commonly used systems. This reflects the downside of using static analysis in measuring "important" or "central" classes in a software system. Using static analysis to measure the centrality of Java libraries is also flawed because the classes are designed more for programmers to use rather than as a system of classes interacting with calls between the classes. For example, the links to java.lang.Object measure references, but in practice calls to methods in java.lang.Object are not actually made. Rather, code would invoke a method of a subclass which overrides a method rather than relying on the default behavior of methods in java.lang.Object.

64. Moreover, the fact that java.lang and related classes dominate the list of supposedly central classes, comprising 30 of what Dr. Kemerer regards as the 32 most central classes, comes as no surprise. The java.lang classes include the standard and basic blocks that any programming language would have, things like strings, integers, and exceptions. The innovations that make software like Android useful and interesting would be built using these

sorts of classes, but all object-oriented computer programming languages would have analogs to just these sorts of classes that would be used to build specific instances of more innovative and useful classes.

65. Dr. Kemerer also conducts what he calls two "sensitivity tests": first seeing what happens if he eliminated the 61 classes that Dr. Reinhold said were necessary for the Java programming language from the mix, and then examining the results when all java.lang classes are removed. In each case, the supposed "centrality" of the remaining classes decreases when these key classes are removed from the data set. Contrary to Dr. Kemerer's assertion that the removal of these classes "has no substantive effect" on the results, this demonstrates the importance of the 61 classes and/or java.lang to the Java programming language, since a decrease in centrality of the remaining classes implies that a substantial portion of the supposed centrality came from these classes that are needed for compatibility with the Java programming language.

66. Dr. Kemerer's appendices setting forth the identity of the supposedly most central classes demonstrates the lack of reliability of his analysis as well. Dr. Kemerer puts java.security.AccessController in the top 75 most central classes in his analysis. However, that class file is specifically noted to be legacy security code that should not be used. Accordingly, a determination that something is "central" in the sense of having a number of connections to other classes does not guarantee that the supposedly central class actually performs any functionality.

67. Dr. Kemerer's analysis of centrality is also contradicted by what Sun/Oracle itself has done with the Java ME platform. Java ME does not contain all of, or even a majority of, the 37 Java SE API packages, and therefore is also missing many of the classes that Dr. Kemerer's analysis claims to be central.

C. Dr. Kemerer's "stability" tests still do not provide an accurate measurement of actual stability

68. As explained in my rebuttal report, Dr. Kemerer's analysis purporting to show when APIs are stable suffers from a glaring flaw: "newly added or deleted methods *are not considered as a change in this analysis*." Kemerer Rebuttal Report ¶ 89 (emphasis added). Because that flaw is still present in his rebuttal report analysis, that analysis is no more meaningful or reliable than the analysis in his opening report.

69. Moreover, his rebuttal report itself identifies evidence of the impropriety of his methodology. He cites a paper upon which he purports to rely for his analysis as saying that "most of the API changes that break the backward compatibility" (*i.e.*, that cause problems for stability of APIs) "are from types of changes including . . . deleted method." Kemerer Rebuttal Report ¶ 81. Dr. Kemerer also cites a paper by Raemakers et al. as purportedly supporting his methodology. Yet the authors of that paper "consider an API to be stable if functionality is not removed from a public interface once it has been added." Thus, even the papers cited by Dr. Kemerer show that a proper methodology would need to take into account the deletion of methods from APIs in order to produce meaningful results. As discussed in my rebuttal report, Android in fact maintains backwards compatibility, such that methods are not deleted in a manner that would cause problems for developers. *See* Astrachan Rebuttal Report ¶ 173 ("Updates to the framework API are designed so that the new API remains compatible with earlier versions of the API. That is, most changes in the API are additive and introduce new or replacement functionality. As parts of the API are upgraded, the older replaced parts are deprecated but are not removed, so that existing applications can still use them.").

70. Dr. Kemerer has now provided the raw data underlying his analysis. From analyzing what that raw data reveals, I have concluded that Dr. Kemerer's analysis of stability

measures changes between levels in Android using a simplistic syntactic analysis of changes between levels when at least some level of semantic analysis would both be more meaningful and correct for syntactic differences that are not meaningful. One of the most striking things about the data is that when Dr. Kemerer claims to find changes, they sometimes come in large bunches of more than one hundred. For example, his raw data identifies 179 alleged changes between API level 21 and API level 22 in android.hardware.camera2. I have investigated the source code for those two levels of the API, and the more than one hundred purported "changes" that Dr. Kemerer reports in that class appear to be spurious. In noting changes between Android levels 21 and 22, there are no changes in the code from the class CaptureRequest in the package android.hardware.camera2; running the command diff shows only comment changes between the Android levels for this class, and no code changes. However, changes in the automatic generation of the current.txt files between API level 21 and 22 causes roughly 50 references to static constants to be identified as different in the class CaptureRequest between the two levels. Below is an example of five of these roughly 50 differences.

71. These are from the API level 21 current.txt file:

```
field public static final android.hardware.camera2.CaptureRequest.Key CONTROL_AF_MODE;
field public static final android.hardware.camera2.CaptureRequest.Key CONTROL_AF_REGIONS;
field public static final android.hardware.camera2.CaptureRequest.Key CONTROL_AF_TRIGGER;
field public static final android.hardware.camera2.CaptureRequest.Key CONTROL_AWB_LOCK;
field public static final android.hardware.camera2.CaptureRequest.Key CONTROL_AWB_MODE;
```

72. In the API level 22 current.txt file, these are "replaced" by the following:

```
field public static final
android.hardware.camera2.CaptureRequest.Key<java.lang.Integer> CONTROL_AF_MODE;
field public static final
android.hardware.camera2.CaptureRequest.Key<android.hardware.camera2.params.MeteringRectangle[]>
CONTROL_AF_REGIONS;
field public static final
android.hardware.camera2.CaptureRequest.Key<java.lang.Integer> CONTROL_AF_TRIGGER;
field public static final
android.hardware.camera2.CaptureRequest.Key<java.lang.Boolean> CONTROL_AWB_LOCK;
field public static final
android.hardware.camera2.CaptureRequest.Key<java.lang.Integer> CONTROL_AWB_MODE;
```

73. The differences shown in these summary files are caused by generic types included in the file for API level 22; that is, the class name between angle brackets such as <java.lang.Integer>. These generic types are part of both source files in API level 21 and API level 22 of Android, but were not printed as generics in the API level 21 current.txt file; hence these don't represent actual changes in the code between the levels. Because there are no code differences, there would be no changes in stability due to what are essentially differences in the documentation produced automatically to summarize the classes.

74. Dr. Kemerer's raw data elsewhere appears to reflect formal changes in the manner in which code is expressed that makes the developer's task easier, but does not result in any difference in the working of existing code. Specifically, he indicates that there are 116 changes between API levels 21 and 22 in the package android.webkit. These changes appear to be modifying a MustOverrideException in a method to instead make the method and the class abstract. However, given the fact that the method previously threw a MustOverrideExeception, a change to abstract would result in no difference in performance observed by a developer.

75. Dr. Kemerer also apparently uses a simplistic syntactic count to indicate that there are 108 changes in the package android.renderscript between API level 15 and API level 16. Between those two API levels, there are exactly 108 changes of public methods and constructors marked as "deprecated," counting across the 49 public classes with source files in the android.renderscript package. Yet such deprecations would not cause any difference in the behavior of code at runtime. By marking methods as deprecated, code already written would continue to run and new code would be flagged at compile time as deprecated, through it would still work. Marking code as deprecated makes the codebase more stable, since programs continue to work and programmers are warned that alternative classes are available as they

develop code. Illogically, under Dr. Kemerer's methodology if Google had chosen to instead delete these methods (causing serious issues with runtime performance), that would not have counted as a change.

76. Dr. Kemerer's raw data reflects slightly more than 2000 supposed changes in the Android framework APIs over all of the different API levels; these several hundred erroneous identifications of changes are a significant portion of what he claims to have found. Given the limited amount of time that I have had access to Dr. Kemerer's raw data, these are only representative errors, and it seems likely that other such errors would exist as well. Correcting these errors would result in the framework APIs showing more "stability," even under Dr. Kemerer's flawed definition of stability.

77. Dr. Kemerer's graphs are also misleading in that they report cumulative changes rather than changes at any given level, thus making later API levels appear to be more unstable when in reality the slope of the line (which would represent Dr. Kemerer's flawed stability measurement) is what matters for purposes of Dr. Kemerer's model. Dr. Kemerer also fails to take into account the relative sizes of the packages in which changes are supposedly found, whether measured by lines of code or numbers of methods contained in the packages. A measure that reflects only the number of changes per package without accounting for the relative size (and change in relative size) of those packages over time is likely to mislead. Accordingly, I believe that Dr. Kemerer's stability analysis is flawed for these reasons, as well as those set forth in my rebuttal report, and does not provide a reliable measure of the stability of Android and Java SE.

78. Because his methodology is flawed, the fact that he also ran it against a data set with either 61 class files or the entirety of java.lang excised from the set provides no greater insight.

79. As noted in my rebuttal report, Dr. Kemerer's conclusions regarding the time at which user activations, number of developers, and number of applications began to grow appears to be based on eyeballing graphs rather than a rigorous analysis. This issue is brought into even starker relief in his rebuttal report, since he now contends that his graphs show that growth began "circa January 2011," Kemerer Rebuttal Report ¶¶ 101, 102, and 104, while in his opening report he contended that the growth began around December 6, 2010, Kemerer opening report ¶¶ 112, 113, and 115.

IX. CONCLUSION

80. I reserve the right to update and refine my opinions and analyses in light of any additional materials or information that may come to my attention in the future, including additional contentions or information produced by Oracle as well as any rulings issued by the Court in this case. I also reserve the right to supplement my opinions and analyses as set forth in this report in light of any expert reports submitted by Oracle and in light of any deposition or trial testimony of Oracle's experts.

Executed on the 29th of February, 2016 in Durham, NC.

Dr. Owen Astrachan

A.Martin, *Google is Forcing Routebuilder to Shut Down* , Medium, https://medium.com/hacker-daily/google-maps-is-forcing-routebuilder-to-shutdown-615ce42f413a#.q467avy9j
Adam Jaffe Rebuttal Report, February 8, 2016
AdWords API Beta: Terms and Conditions, Google Developers, http://web.archive.org/web/20050130011554/http://www.google.com/apis/adwords/terms.html
AdWords API Beta Terms and Conditions, Google AdWords, http://web.archive.org/web/20050130011554/http://www.google.com/apis/adwords/terms.html
AdWords API Terms and Conditions, Google AdWords, http://web.archive.org/web/20050130011554//http://www.google.com/apis/adwords/terms.html
AdWords API Terms and Conditions, Google AdWords, https://web.archive.org/web/20140704175508/https://developers.google.com/adwords/api/docs/terms
AdWords API Terms and Conditions, Google AdWords, https://web.archive.org/web/20150406213931/https://developers.google.com/adwords/api/docs/terms
AdWords API Terms and Conditions, Google AdWords, https://web.archive.org/web/20150922081038/https://support.google.com/adwordspolicy/answer/6169371
AdWords API Terms and Conditions, Google Developers, https://developers.google.com/adwords/api/docs/terms?hl=en
B. Darrow, *Oracle v. Google ruling shows why cloud players may have steered clear of Amazon API,* https://gigaom.com/2014/05/09/oracle-v-google-ruling-shows-why-cloud-players-may-have-steered-clear-of-amazon-apis
Bing Search API , Microsoft Azure Marketplace, http://datamarket.azure.com/dataset/bing/search#terms
Chris Kemerer Rebuttal Report and appendices, February 8, 2016
Chris Kemerer Underlying Data
Configuration and Reporting API Limits and Quotas, Google Developers, https://developers.google.com/analytics/devguides/reporting/mcf/v3/limits-quotas
Deposition of Bob Lee, August 3, 2011
Deposition of Urs Holzle, November 24, 2015
Dig, D. and R. Johnson, *The Role of Refactorings in API Evolution,* Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM '05), pp. 389–398, Washington, DC, USA, 2005, IEEE Computer Society.
Douglas Schmidt Rebuttal Report and appendices, February 8, 2016
Douglas Schmidt Rebuttal Report correted appendices, February 18, 2016
FAQ for the Bing Search API: Windows Azure Marketplace, Microsoft Corp., https://onedrive.live.com/view.aspx?resid=9C9479871FBFA822!110&app=Word&authkey=!AInKSlZ6KEzFE8k
Google APIs Terms of Service, Google Developers, https://developers.google.com/terms
Google is No Longer Forcing RouteBuilder to Shut Down , Medium, https://medium.com/@andrewcmartin/google-is-no-longer-forcing-routebuilder-to-shut-down-8d4882f31447#.4jlgsvmu8
Google Maps APIs Pricing and Plans, Google Developers, https://developers.google.com/maps/pricing-and-plans/?hl=en

Google Maps/Google Earth APIs Terms of Service , Google Developers, https://developers.google.com/maps/terms?hl=en
Google Maps/Google Earth APIs Terms of Service, Google, https://developers.google.com/maps/terms
https://developers.google.com/maps/pricing-and-plans/#details
ICU - International Components for Unicode at http://site.icu-project.org
Ioana Sora, *Helping Program Comprehension of Large Software Systems by Identifying Their Most Important Class,* Department of Computer and Software Engineering University Politehnica of Timisoara, Romania, EnaseSoraSpringer2015, http://staff.cs.upt.ro/~ioana/papers/EnaseSoraSpringer2015.pdf
Jasper S20 Cellphone Rocks The Java OS, People Flee in Fear, http://gizmodo.com/210575/jasper-s20-cellphone-rocks-the-java-os-people-flee-in-fear.
Java 2 Platform Standard Edition Development Kit 5.0 Specification, Oracle, http://docs.oracle.com/javase/1.5.0/docs/relnotes/license.html
McDonnell, T. B. Ray and M. Kim. *An Empirical Study of API Stability and Adoption in the Android Ecosystem,* 29th IEEE International Conference on Software Maintenance (ICSM), pp.70-79, 22-28 Sept. 2013. doi: 10.1109/ICSM.2013.18
P. Ranade, D. Scannell and B. Stafford, *Ready for APIs? Three steps to unlock the data economy's most promising channel,* Forbes, Jan. 7, 2014, http://www.forbes.com/sites/mckinsey/2014/01/07/ready-for-apis-three-steps-to-unlock-the-data-economys-most-promising-channel/#51a1d71e89e5
Prediction API Pricing, Google Cloud, https://cloud.google.com/prediction/#pricing
Raemaekers, S., A. van Deursen, and J. Visser. *Measuring software library stability through historical version analysis,* 28th IEEE International Conference on Software Maintenance (ICSM), pp.378-387, Sept. 23-28 2012. doi: 10.1109/ICSM.2012.6405296
SavaJe releases Jasper S20 Java phone, http://www.engadget.com/2006/05/13/savaje-releases-jasper-s20-java-phone
Terms of Service, Google Cloud Platform, https://cloud.google.com/translate/v2/terms
Translate API Pricing , Google Cloud, https://cloud.google.com/translate/v2/pricing
TX 1028
TX 1062
Xcode and Apple SDKs Agreement, Apple Inc., https://www.apple.com/legal/sla/docs/xcode.pdf